by Jeff Stefan

Fuzzy Logic

This article explains how fuzzy logic works and provides you with a simple program so you can experiment with fuzzy logic systems of your own.

The Beginnings

Western thought is fundamentally based on Aristotelian logic and the "Law of the Excluded Middle." Either something is true or false, on or off, while everything in the middle is ignored.

According to the classical Western view of the universe, things aren't partially on or off, or partially true or false. Formal logic is based on this outlook.

This same view worked its way into abstract mathematics, beginning with set theory as invented and explored in the late 1800s by the mathematician George Cantor.

Although there's no formal definition of a set, the concept of sets is familiar to most of us. A set is a collection of objects. The only required property of a set object is membership; either something's in the set or it's not. One set all by itself isn't worth much, so how do multiple sets interact?

They interact via three fundamental set operations: Union, Intersection, and Complementation. See Figure 1 for a listing of fundamental set operations.

The Aristotelian viewpoint has propagated like a virus from logic to set theory, and mutated with the evolution of Boolean algebra. The names for intersection, union, and complementation change from set theory, formal logic, and Boolean logic, but the meaning is the same.

See Figure 2 for a list illustrating the names for the same operations in each of the domains, including fuzzy sets (the Fuzzy Logic operations will be explained later).

Boolean logic turned out to be a nearly perfect tool to describe switching and logic circuits. The digital computer is basically an Aristotelian logic device, and inherently Western in concept.

All this translates to information loss when describing the real world to a computer. What about the truths that reside in the excluded middle? We know from our everyday experience that the universe does not present itself in clear-cut, black and white, yes or no configurations. Many things are somewhat true or somewhat false. The universe presents you with the option of choosing some degree of truth or falsehood. How do we reflect these choices within a computer program?

Another big problem is the subjective nature of words. How do you map words and phrases into numbers? Phrases like "a little" or "a lot" are hard to quantify. How do you represent the weight of words in a computer program?

The problem of quantifying middle truths did not escape the thoughts of some prominent logicians throughout recent history. The Polish philosopher and logician Jan Lukasiewicz described a logical system in 1920 that consisted of three values: zero (false), one (true), and one-half (possible). Cornell professor Max Block described an early model of fuzzy sets in his 1937 paper entitled "Vagueness: An Exercise in Logical Analysis."

All of this work solidified in 1965 with the publication of Stanford Professor Lotfi Zadeh's paper entitled "Fuzzy Sets," in the June 1965 issue of the journal *Information and Control*. In an early pre-publication presentation of his paper, Zadeh was faced with severe criticism. This criticism dogged fuzzy logic for years, leading to a lack of research grants.

No one wanted to fund research for a risky, potential loser like fuzzy logic, and many claimed that fuzzy logic was just probability theory in disguise. If you can't get the money to study an emerging technology's properties and potential applications, then the new technology stalls and drifts to the unfunded, obscure backwaters of academia.

No one in the United States took fuzzy logic seriously, but things were different in Japan. Unburdened by the weight of Western Aristotelian Logic, the Japanese embraced fuzzy logic. Instead of arguing over the theoretical merits and faults of fuzzy logic, the Japanese produced fuzzy-based systems. Some of the Japanese fuzzy-based systems in production today are single button washing machines, subway and elevator controls, voice and handwriting recognition systems, and auto-focus cameras.

Basic Principles

There are three basic components to a fuzzy system: the fuzzifier, the rule base, and the de-fuzzifier, as shown in Figure 3.

Inputs enter the fuzzifier and are assigned a numerical value ranging from zero to one, depending on the input value's "degree of truth." Numbers themselves can be fuzzy. Say you're meeting someone for lunch at "around 12 o'clock." Fuzzy logic allows you to quantify vague situations like this.

We can assign truth values to "around 12 o'clock" as illustrated in Figure 4.

Truth values range from 0.0 to 1.0, with zero indicating "completely false" and one indicating "completely true." This part corresponds exactly to Boolean logic.

The excluded middle area is assigned numerical values. As the time values on the X axis approach 12:00 p.m., the truth values on the Y axis approach one. After 12:00 p.m., the truth values on the Y axis decrease.

By assigning a degree of truth value to time, we know that 11:45 a.m. has a truth value of 0.5, which tells us that 11:45 a.m. is "somewhat true" in terms of being close to our lunch date. As a rough guide to the meaning associated with truth values, see Figure 5. Just remember that the meanings associated to the truth values are arbitrary.

The linguistic meaning attached to the truth values are dependent on the system inputs you are

Fundamental Set Operations

Union	*- contains all of the objects in set A and set B*
Intersection	*- contains only the objects that are in set A and set B*
Complement	*- set of all objects not in a set*

Fig. 1

Set Theory	Formal Logic	Boolean Logic	Fuzzy Logic
Intersection	Conjunction	AND	Min(x,y)
Union	Disjunction	OR	Max(x,y)
Complementation	Negation	NOT	1.0 - x

Fig. 2

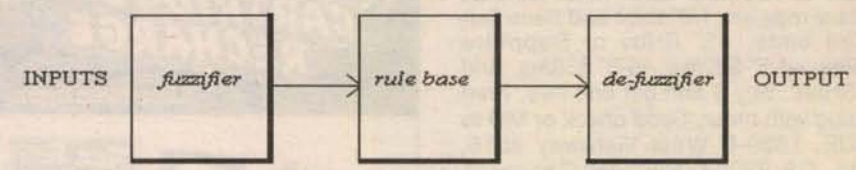

Fig. 3

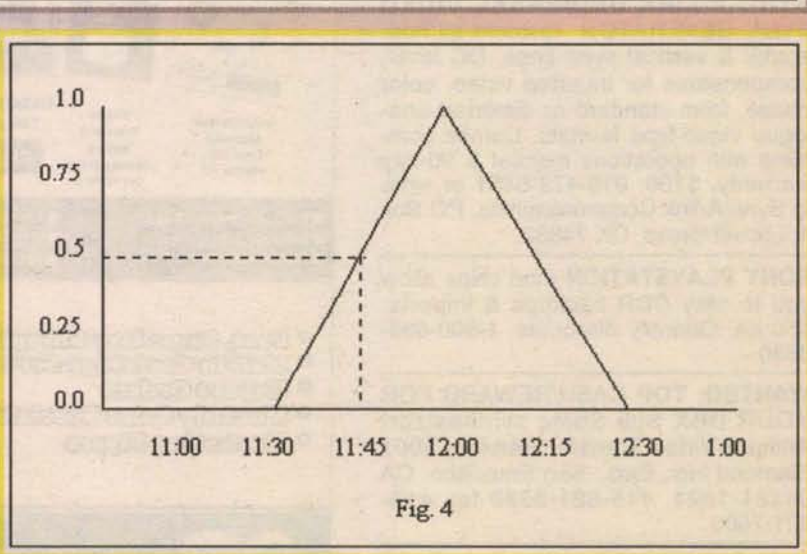

Fig. 4

fuzzifying. If you're looking at temperature, and if you consider 90 degrees F to be hot, then you could assign a truth value of .83, which is close to "mostly true."

Lost in Space

Let's say we're controlling a spacecraft re-entry vehicle, and we need to monitor velocity and heat shield temperature. If the heat shield gets too hot, then we have to change our angle of attack or else burn up. If we go in at too steep an angle, then we'll miss our landing coordinates. If we go too in at too shallow an angle, we'll skip off the atmosphere like a flat stone on water and spin off into the black regions of space. We intend to build a fuzzy control module that will keep the spacecraft within nominal limits.

The first thing we need to do is to assign degrees of truth values to the sensor input values, which are the temperature and re-entry angle of the spacecraft. These are called hedges. The graphs of these values are shown in Figure 6 and Figure 7. We have hedge values for both temperature and angle.

Once the input values are mapped to Y axis truth values, a rule base to manipulate the values must be established. We have eight rules in our system, which are listed in Figure 8.

These rules are of IF-THEN form, and are similar to rules found in expert systems, only they sound much more natural with the addition of the modifier "slightly." That's the beauty of Fuzzy Logic: we can use seemingly imprecise words like "slightly" or "somewhat," and assign them a precise numerical value.

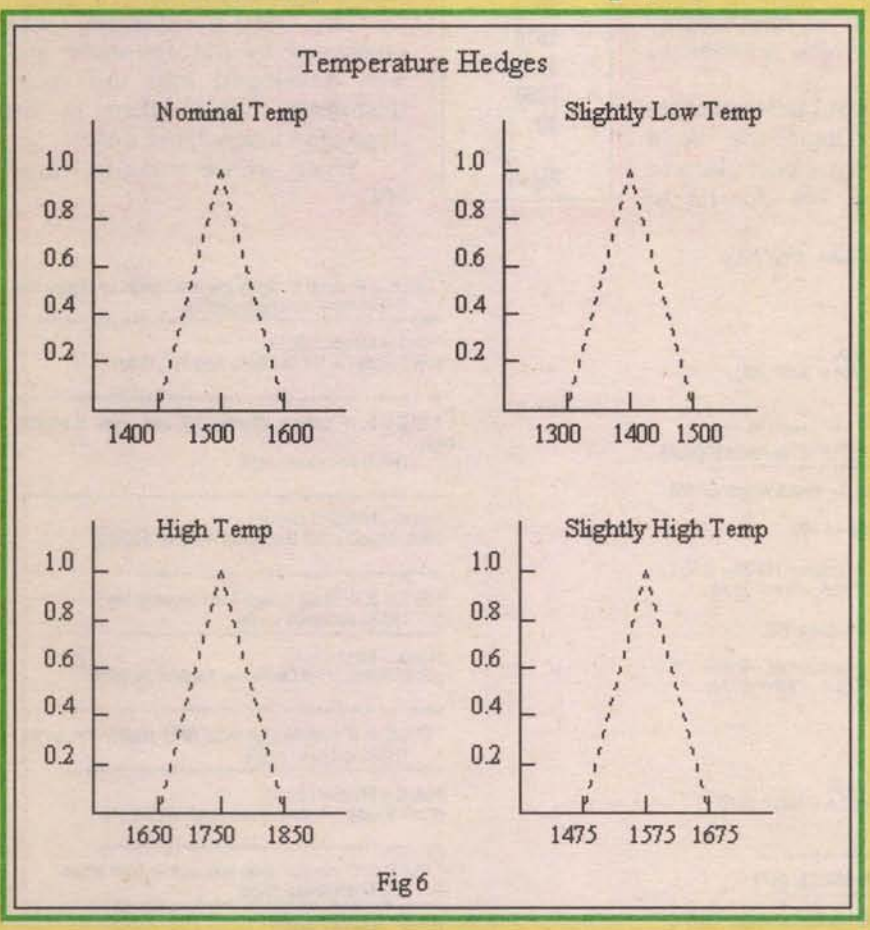

Fig 6

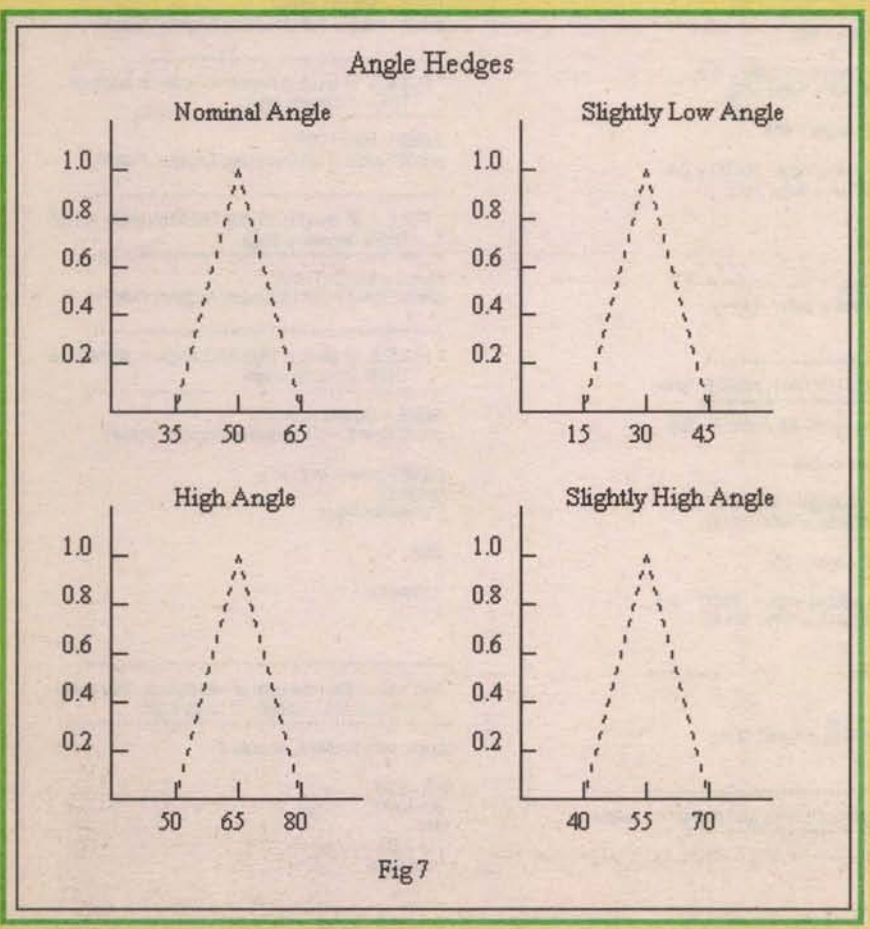

Fig 7

Determining a Range Function

We need two functions to fuzzify the incoming temperature and angle values. Here's one method: First, find the difference between the high and low value for the hedge. Looking at the Nominal Temp graph in Figure 6, the high value is 1600 and the low value is 1400. The difference is 200. We need a two-sided equation, since our hedge values rise on one side of the nominal value and fall on the other. The following is the pseudo-code instructions for the Nominal Temp graph:

```
IF Temp >= 1400 && Temp < 1600 THEN
 IF Temp <= 1500 THEN
   FuzzyTemp = (Temp - 1400)/200 + 0.5
ELSE IF Temp > 1500 THEN
 FuzzyTemp = fabs(Temp -1600)/200 + 0.5
ELSE
 FuzzyTemp = 0
```

If the temperature value is in the range we're looking for, and is less than or equal to the nominal temperature — 1500 — then subtract the lower limit from the temperature value and divide by the difference between the high and low values, which is 200. Add 0.5 to the result to scale it properly to the Y axis.

In order to get decreasing values for temperatures greater than the nominal value of 1500, subtract the temperature reading from the high value, and again divide by the range and add 0.5.

We need to take the absolute value of the result, since all we care about is the truth value and not its sign. This same method is used for all of the graphs for temperature and angle hedges. Again, this is just one of many ways to assign truth values. Assigning truth values is completely arbitrary, and the method used here makes sense for our system, but may not for another.

Coding the rule base is very easy. To AND two fuzzy numbers together, all we need to do is to find the smaller of the two numbers. Rule 1 is coded as:

```
Rule1 = Min(SLT,SLA);
```

where SLT stands for Slightly Low Temp and SLA stands for Slightly Low Angle. The Min function itself is simple:

```
double Min(double A, double B)
{
 if (A <= B)
 return(A);
      else
      if (A > B)
       return(B);
      }
```

We now have a rule base and our membership functions defined. What do we do with them? We read the inputs, run them through the fuzzification functions and the rules, then set an output to take some kind of action. In our case, we need to adjust the angle of the re-entry vehicle to avoid burning up. Here's where you, as the designer, must make some significant design choices.

You can de-fuzzify the outputs from the rule base and use the crisp value as a trigger for some control mechanism, or you can choose not to de-fuzzify the outputs.

In our example, the outputs from the rule base are printed and not de-fuzzified. In order to correct the angle of the re-entry vehicle in our system, take the highest numerical value produced by the rules and use that value as truth. A sample run for a temperature-angle pair is shown in Figure 9. The data

Truth Value	Meaning
0.0	Completely false
0.2	Mostly false
0.4	Somewhat false
0.6	Somewhat true
0.8	Mostly true
1.0	Completely true

Fig. 5

Rule Base

RULE 1: IF temp is slightly low AND angle is slightly low THEN increase angle.

RULE 2: IF temp is slightly high AND angle is slightly high THEN decrease angle

RULE 3: IF temp is high AND angle is high THEN decrease angle

RULE 4: IF temp is slightly high AND angle is slightly low THEN increase angle

RULE 5: IF temp is nominal AND angle is slightly high THEN decrease angle

RULE 6: If temp is high AND angle is nominal THEN decrease angle

RULE 7: IF temp is slightly low AND angle is high THEN decrease angle

RULE 8: IF temp is high AND angle is slightly low THEN increase angle

Fig. 8

file used with the program is called fval.dat and is listed in Figure 10.

Any data files you use for the program need to be in this format. The first value is the current temperature read from the sensors, and the second value is the re-entry angle. The program is named fuzz.exe, and the data file name in input from the command line. Just type: fuzz fval.dat to get the program to run.

In the trial run shown in Figure 9, the nominal angle (NA) value is 1.0, which is completely true. The temperature is high (HT) with a value of .95, so looking at the highest value in the rule base, it tells us via Rule 6 to decrease the re-entry angle. This is what we expect the rule base to tell us.

```
---------[READ INPUTS]----------
Temp = 1760.000000
Angle = 50.000000
---------[FUZZIFICATION]----------
NT = 0.000000
SLT = 0.000000
HT = 0.950000
SHT = 0.000000
NA = 1.000000
SLA = 0.000000
HA = 0.500000
SHA = 0.833333
---------[RULE BASE OUTPUT]----------
Rule1 = 0.000000 Increase Angle
Rule2 = 0.000000 Decrease Angle
Rule3 = 0.500000 Decrease Angle
Rule4 = 0.000000 Increase Angle
Rule5 = 0.000000 Decrease Angle
Rule6 = 0.950000 Decrease Angle
Rule7 = 0.000000 Decrease Angle
Rule8 = 0.000000 Increase Angle
```

Fig. 9

If you wanted just to look at one single value as the output of the rule base, then take all of the outputs and OR them together. In fuzzy logic, an OR operation means taking the maximum value of a set of numbers.

As usual, the code is yours to play with. I used Turbo C/C++ 3.0 for the example, but any C compiler should work equally as well. Adding an OR function to create a single output number from the set of rule outputs would be a nice addition to the code.

If you want to learn more about fuzzy logic, the World Wide Web is the best place to start. There are dozens of fuzzy resources on the Web so you can easily expand your personal knowledge base if your interest is serious.

It's strange how history repeats itself. In 1937, the same year that Max Block's published his early paper on fuzzy sets, Howard Hughes approached the US Army about a new, lightning quick plane he recently designed and built, the H-1.

FVAL.DAT

```
1450
30
1325
60
1475
50
1650
24
1760
50
1650
35
1800
20
```

Fig 10

The H-1 was highly maneuverable and flew in excess of 330 MPH — faster than any American airplane built. Just like Zadeh's notion of fuzzy sets, Hughes's H-1 was rejected by the Americans. Guess what happened?

The H-1's design was embraced by the Japanese and was developed into the most destructive fighter plane in the Japanese arsenal, the Zero.

When are we going to learn?

NV

```
/*****************************************************
* Filename: fuzz.c
* Description: Fuzzy Logic demo.
*
*
* (c) 1998 Jeff Stefan
*****************************************************/
#include <stdio.h>
#include <math.h>

double Min(double A, double B);
#define MAX_FILE_NAME 45

void main(int argc, char *argv[])
{
FILE *fp;
char Filename[MAX_FILE_NAME];
int Result;
double Temp,Angle;
double NT,SLT,HT,SHT,NA,SLA,HA,SHA;
d o u b l e
Rule1,Rule2,Rule3,Rule4,Rule5,Rule6,Rule7,Rule8;

 /*********************************
 * get filename from command line
 *********************************/
 if(argc < 2)
 {
  printf("Usage: fuzz filename\n");
  exit(0);
 }

 /***********
 * open file
 ***********/
 strcpy(Filename,argv[1]);
 fp = fopen(Filename,"r");
 if(fp == NULL)
 {
  printf("Can't open file\n");
  exit(0);
 }

 /*****************
 * init variables
 *****************/
 NT=SLT=HT=SHT=0.0;
 NA=SLA=HA=SHA=0.0;

 /*************************
 * read and process inputs
 *************************/
 do
 {
  printf("----------[READ INPUTS]----------\n");
  Result = fscanf(fp,"%lf",&Temp);
  if(Result == -1)
   break;
  printf("Temp = %lf\n",Temp);
  fscanf(fp,"%lf",&Angle);
  printf("Angle = %lf\n",Angle);

  printf("----------[FUZZIFICATION]----------\n");
  /*******************
  * NOMINAL TEMP (NT)
  *******************/
  if(Temp >= 1400 && Temp <= 1600)
  {
   if(Temp <= 1500)
   {
    NT = (Temp - 1400)/200 + 0.5;
    printf("NT = %lf\n",NT);
   }
   else if(Temp > 1500)
   {
    NT = fabs(Temp - 1600)/200 + 0.5;
    printf("NT = %lf\n",NT);
   }
  }
  else
  {
   NT = 0.0;
   printf("NT = %lf\n",NT);
  }

  /*************************
  * SLIGHTLY LOW TEMP (SLT)
  *************************/
  if(Temp >= 1300 && Temp <=1500)
  {
   if(Temp <= 1400)
   {
    SLT = (Temp - 1300)/200 + 0.5;
    printf("SLT = %lf\n",SLT);
   }
   else if(Temp > 1400)
   {
    SLT = fabs(Temp - 1500)/200 + 0.5;
    printf("SLT = %lf\n",SLT);
   }
  }
  else
  {
   SLT = 0.0;
   printf("SLT = %lf\n",SLT);
  }

  /****************
  * HIGH TEMP (HT)
  ****************/
  if(Temp >= 1650 && Temp <= 1850)
  {
   if(Temp <= 1750)
   {
    HT = (Temp - 1650)/200 + 0.5;
    printf("HT = %lf\n",HT);
   }
   else if(Temp > 1750)
   {
    HT = fabs(Temp - 1850)/200 + 0.5;
    printf("HT = %lf\n",HT);
   }
  }
  else
  {
   HT = 0.0;
   printf("HT = %lf\n",HT);
  }

  /*************************
  * SLIGHTLY HIGH TEMP (SHT)
  *************************/
  if(Temp >= 1475 && Temp <= 1675)
  {
   if(Temp <= 1575)
   {
    SHT = (Temp - 1475)/200 + 0.5;
    printf("SHT = %lf\n",SHT);
   }
   else if(Temp > 1575)
   {
    SHT = fabs(Temp - 1675)/200 + 0.5;
    printf("SHT = %lf\n",SHT);
   }
  }
  else
  {
   SHT = 0.0;
   printf("SHT = %lf\n",SHT);
  }

  /*******************
  * NOMINAL ANGLE (NA)
  *******************/
  if(Angle >= 35 && Angle <= 65)
  {
   if(Angle <= 50)
   {
    NA = (Angle - 35)/30 + 0.5;
    printf("NA = %lf\n",NA);
   }
   else if(Angle > 50)
   {
    NA = fabs(Angle - 65)/30 + 0.5;
    printf("NA = %lf\n",NA);
   }
  }
  else
  {
   NA = 0.0;
   printf("NA = %lf\n",NA);
  }

  /*************************
  * SLIGHTLY LOW ANGLE (SLA)
  *************************/
  if(Angle >= 15 && Angle <= 45)
  {
   if(Angle <= 30)
   {
    SLA = (Angle - 15)/30 + 0.5;
    printf("SLA = %lf\n",SLA);
   }
   else if(Angle > 30)
   {
    SLA = fabs(Angle - 45)/30 + 0.5;
    printf("SLA = %lf\n",SLA);
   }
  }
  else
  {
   SLA = 0.0;
   printf("SLA = %lf\n",SLA);
  }

  /*****************
  * HIGH ANGLE (HA)
  *****************/
  if(Angle >= 50 && Angle <= 80)
  {
   if(Angle <= 65)
   {
    HA = (Angle - 50)/30 + 0.5;
    printf("HA = %lf\n",HA);
   }
   else if(Angle > 65)
   {
    HA = fabs(Angle - 80)/30 + 0.5;
    printf("HA = %lf\n",HA);
   }
  }
  else
  {
   HA = 0.0;
   printf("HA = %lf\n",HA);
  }

  /*************************
  /* SLIGHTLY HIGH ANGLE (SHA)
  *************************/
  if(Angle >= 40 && Angle <= 70)
  {
   if(Angle <= 55)
   {
    SHA = (Angle - 40)/30 + 0.5;
    printf("SHA = %lf\n",SHA);
   }
   else if(Angle > 55)
   {
    SHA = fabs(Angle - 70)/30 + 0.5;
    printf("SHA = %lf\n",SHA);
   }
  }
  else
  {
   SHA = 0.0;
   printf("SHA = %lf\n",SHA);
  }

  /*****************************************
  * Run through rules and determine output
  *****************************************/
  printf("----------[RULE BASE OUTPUT]----------
\n");

  /*****************************************************
   * RULE 1: IF temp is slighly low AND angle is slightly low
   *     THEN increase angle slightly.
   *****************************************************/
   Rule1 = Min(SLT,SLA);
   printf("Rule1 = %lf Increase Angle\n",Rule1);

   /*****************************************************
    * RULE 2: IF temp is slightly high and angle is slightly
high
    *     THEN decrease angle.
   *****************************************************/
   Rule2 = Min(SHT,SHA);
   printf("Rule2 = %lf Decrease Angle\n",Rule2);

   /*****************************************
   * RULE 3: IF temp is high AND angle is high
   *     THEN decrease angle.
   *****************************************/
   Rule3 = Min(HT,HA);
   printf("Rule3 = %lf Decrease Angle\n",Rule3);

   /*****************************************************
   * RULE 4: IF slightly high temp AND slightly low angle
   *     THEN increase angle
   *****************************************************/
   Rule4 = Min(SHT,SLA);
   printf("Rule4 = %lf Increase Angle\n",Rule4);

   /*****************************************************
   * RULE 5: IF normal temp and slightly high angle
   *     THEN decrease angle
   *****************************************************/
   Rule5 = Min(NT,SHA);
   printf("Rule5 = %lf Decrease Angle\n",Rule5);

   /*****************************************
   * RULE 6: IF temp is high AND angle is nominal
   *     THEN decrease angle.
   *****************************************/
   Rule6 = Min(HT,NA);
   printf("Rule6 = %lf Decrease Angle\n",Rule6);

   /*****************************************************
   * RULE 7: IF temp is slightly low AND angle is high
   *     THEN decrease angle.
   *****************************************************/
   Rule7 = Min(SLT,HA);
   printf("Rule7 = %lf Decrease Angle\n",Rule7);

   /*****************************************************
   * RULE 8: IF temp is high AND angle is slightly low
   *     THEN increase angle.
   *****************************************************/
   Rule8 = Min(HT,SLA);
   printf("Rule8 = %lf Increase Angle\n",Rule8);

   printf("Press a key...\n");
   getchar();
  } while(!feof(fp));

  if(fp)
  {
   fclose(fp);
  }
}

/*****************************************************
* Min: return the minimum of two values. This is the
*   traditional AND function in fuzzy logic.
*****************************************************/
double Min(double A, double B)
{
 if(A <= B)
  return(A);
 else
  if(A > B)
   return(B);
}
```